EBook Gratis

APRENDIZAJE pyqt5

Free unaffiliated eBook created from
Stack Overflow contributors.

#pyqt5

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: pyqt5

It is an unofficial and free pyqt5 ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official pyqt5.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con pyqt5

Observaciones

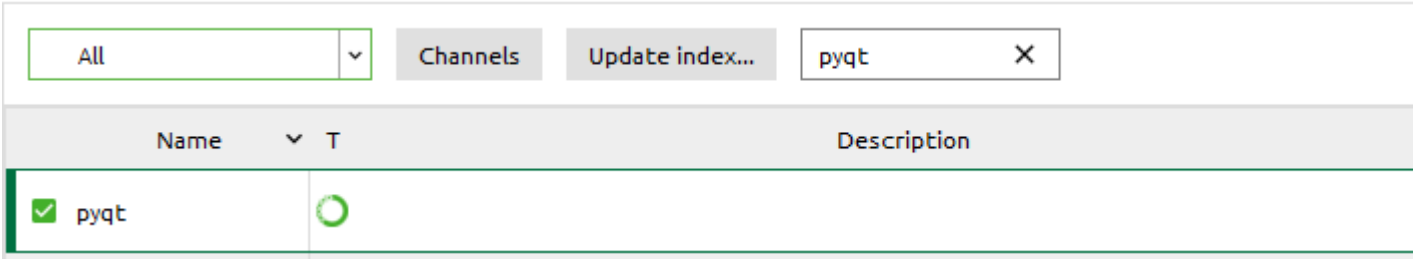
Esta sección proporciona una descripción general de qué es pyqt5, y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de pyqt5, y vincular a los temas relacionados. Dado que la Documentación para pyqt5 es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

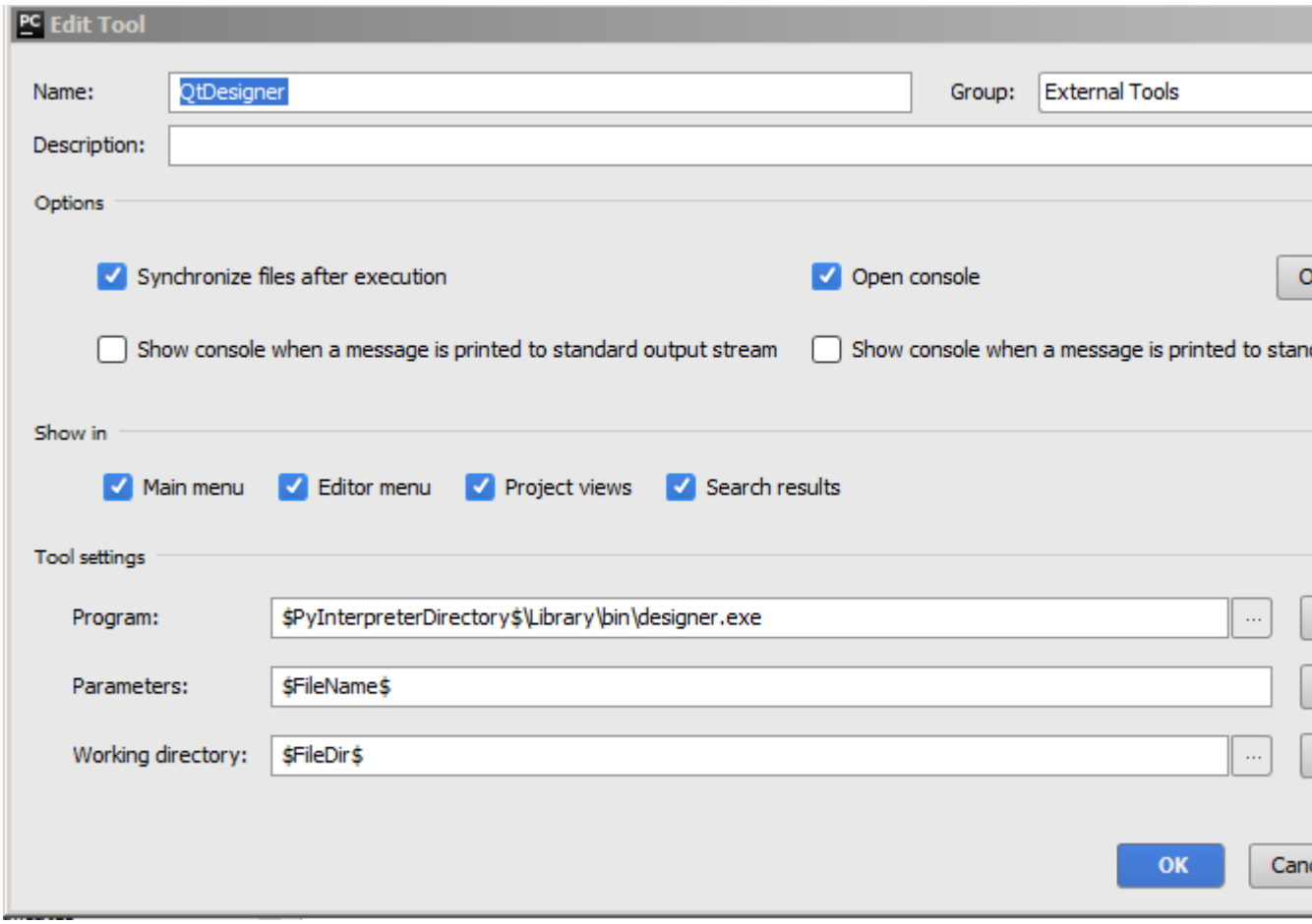
Examples

Instalación o configuración

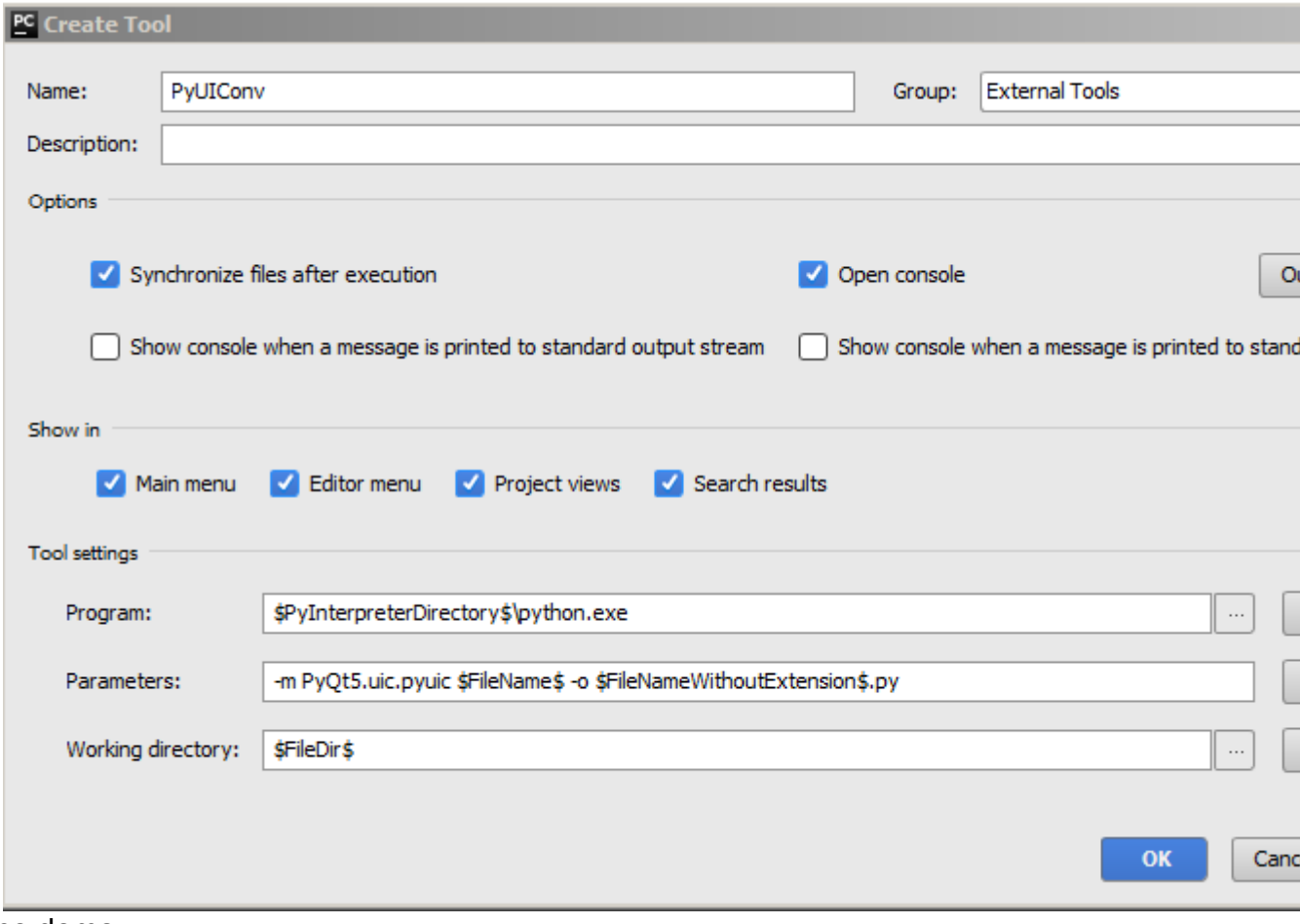
1. Instale Anaconda (PyQt5 es incorporado), especialmente para usuarios de Windows.



2. Integrar QtDesigner y QtUIConvert en PyCharm (herramientas externas)
    - Abra `Settings` PyCharm> `Tools` > `External Tools`
    - Crear herramienta (QtDesigner): se utiliza para editar archivos * .ui

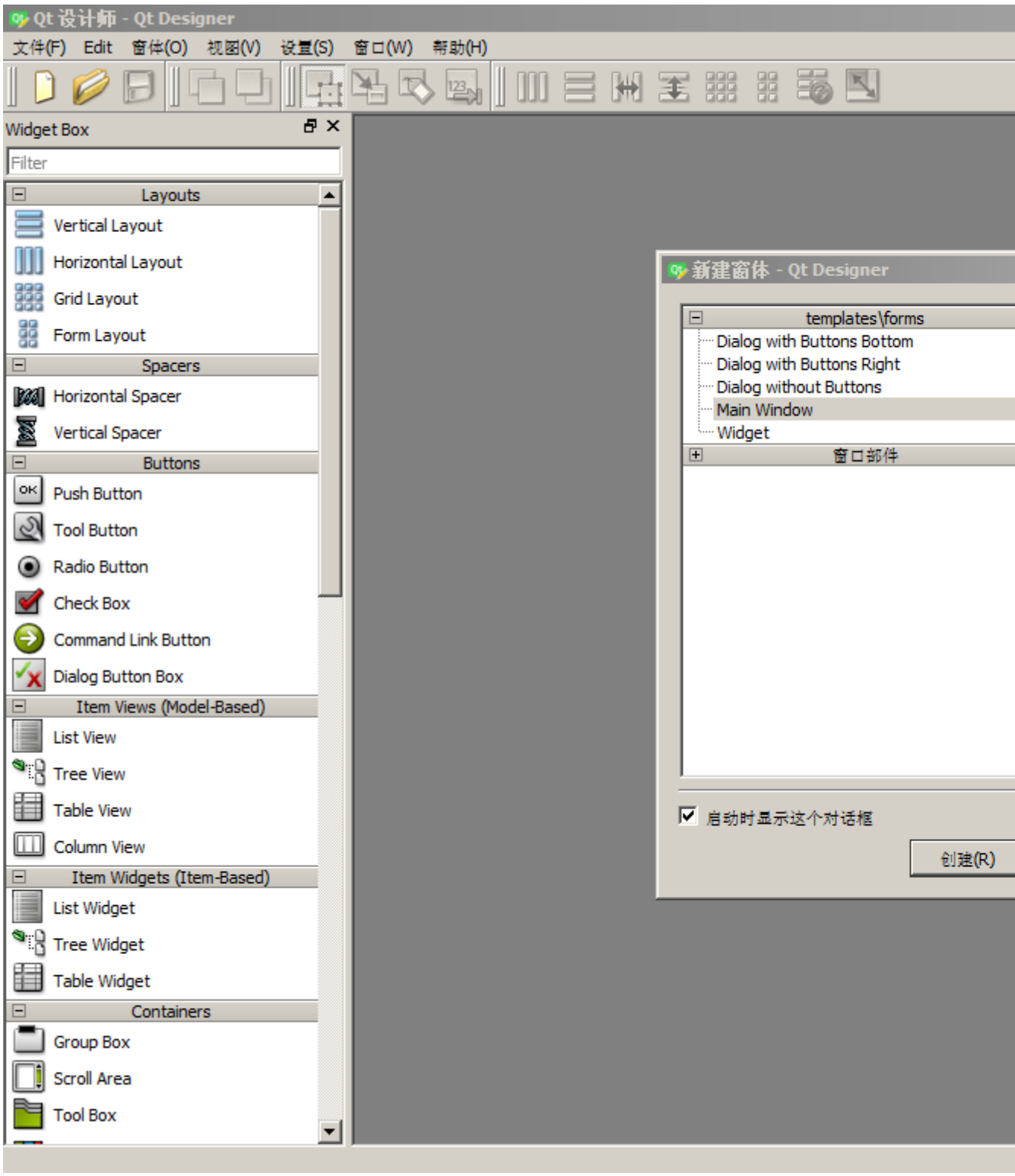


- Crear herramienta (PyUIConv): se utiliza para convertir * .ui a * .py

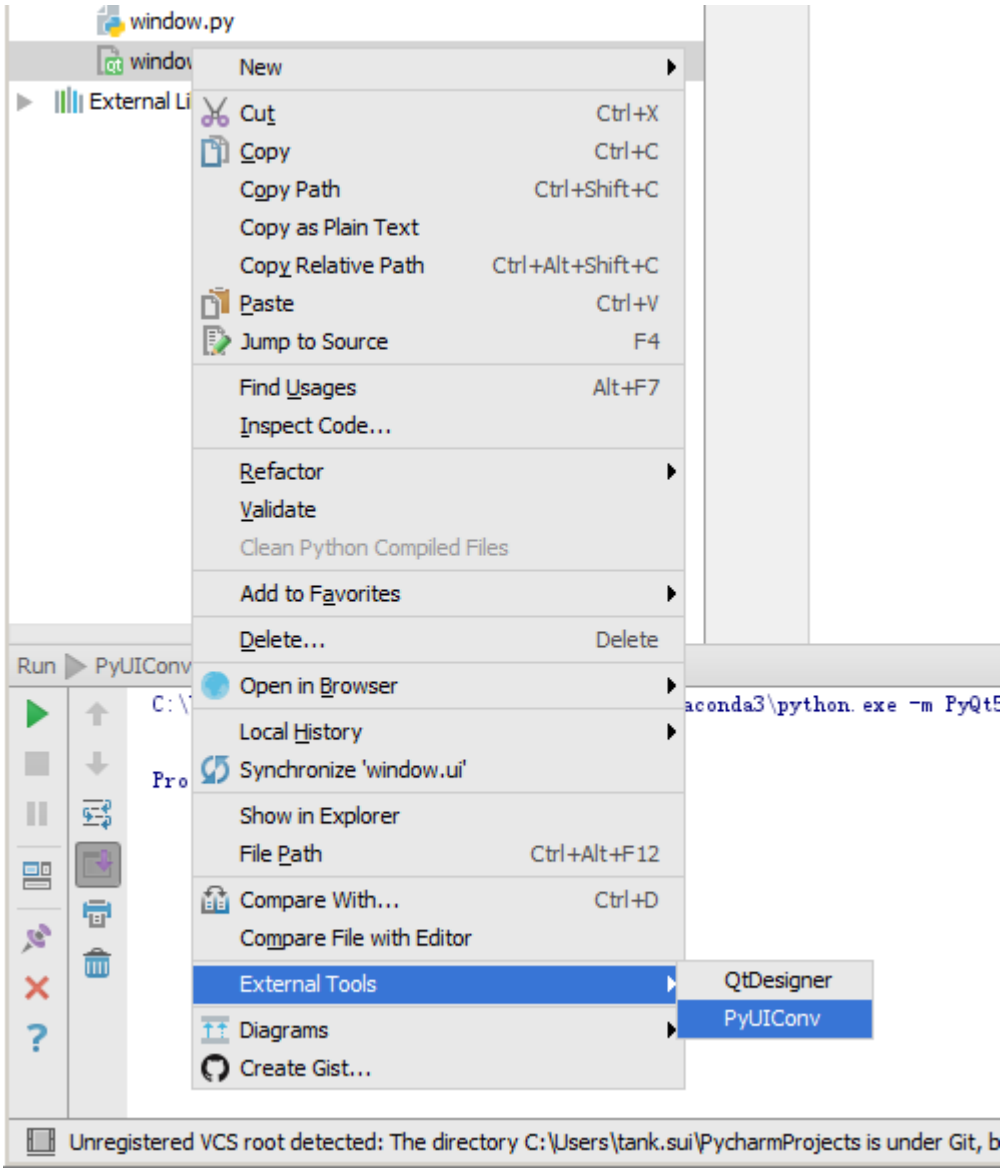


3. Escribe demo

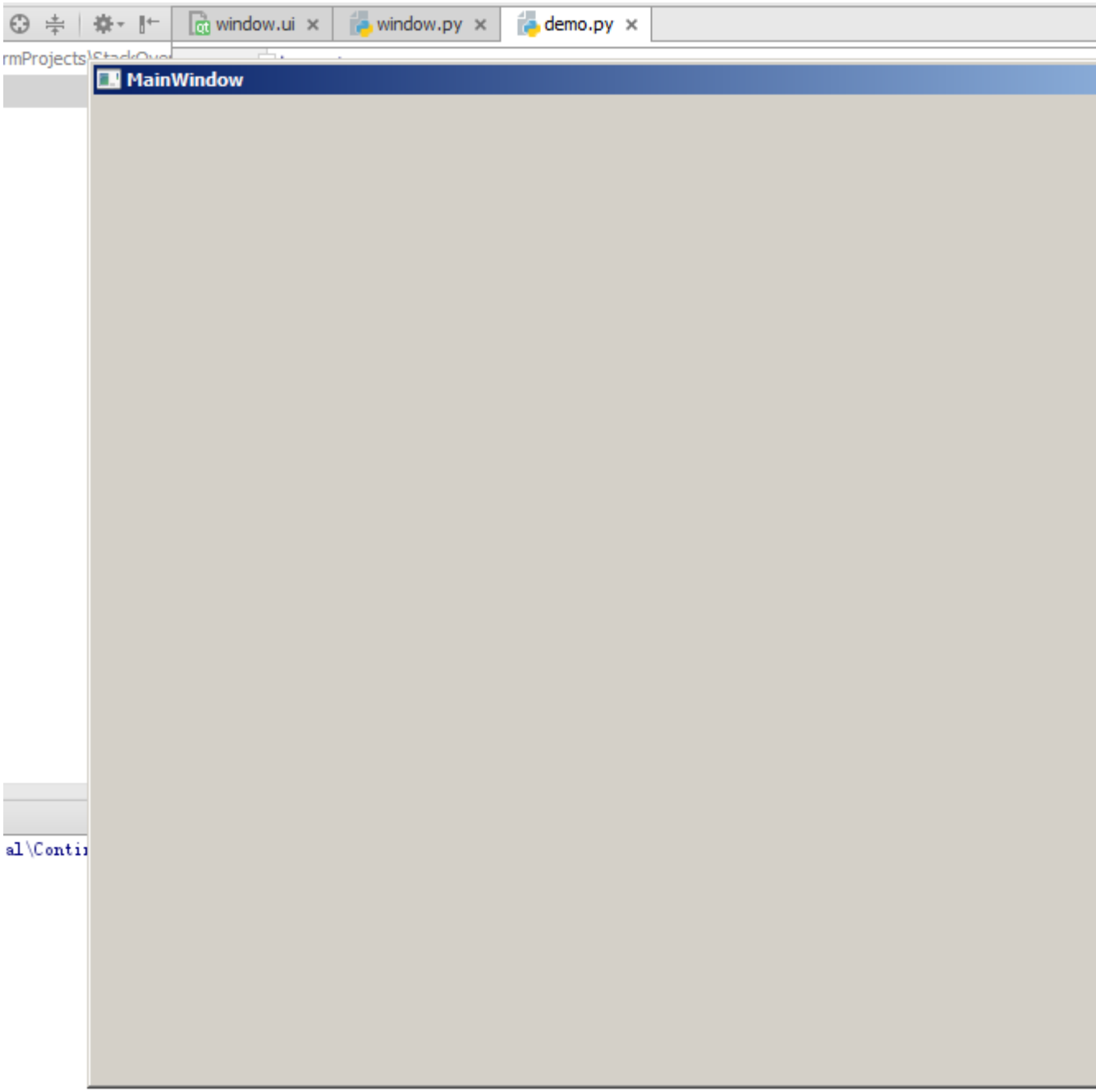
- new window.ui por herramienta externa (QtDesigner)



- convertir a window.py por una herramienta externa (PyUIConv)



- manifestación




```
import sys
from PyQt5.QtWidgets import QApplication,QMainWindow
from window import Ui_MainWindow

if __name__ == '__main__':
    app = QApplication(sys.argv)
    w = QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(w)
    w.show()
    sys.exit(app.exec_())
```

Hola mundo ejemplo

Este ejemplo crea una ventana simple con un botón y una edición de línea en un diseño. También muestra cómo conectar una señal a una ranura, de modo que al hacer clic en el botón se agrega

algo de texto a la edición de línea.

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget


if __name__ == '__main__':

    app = QApplication(sys.argv)

    w = QWidget()
    w.resize(250, 150)
    w.move(300, 300)
    w.setWindowTitle('Hello World')
    w.show()

    sys.exit(app.exec_())
```

Análisis

`app = QtWidgets.QApplication(sys.argv)`

Cada aplicación PyQt5 debe crear un objeto de aplicación. El parámetro sys.argv es una lista de argumentos de una línea de comando. Los scripts de Python se pueden ejecutar desde el shell.

```
w = QWidget()
```

El widget `QWidget` es la clase base de todos los objetos de la interfaz de usuario en PyQt5. Proporcionamos el constructor predeterminado para `QWidget` . El constructor por defecto no tiene padre. Un widget sin padre se llama ventana.

```
w.resize(250, 150)
```

El método resize `resize()` cambia el tamaño del widget. Tiene 250px de ancho y 150px de alto.

```
w.move(300, 300)
```

El método `move()` mueve el widget a una posición en la pantalla en x = 300, y = 300 coordenadas.

```
w.setWindowTitle('Hello World')
```

Aquí ponemos el título a nuestra ventana. El título se muestra en la barra de título.

```
w.show()
```

El método `show()` muestra el widget en la pantalla. Un widget se crea primero en la memoria y luego se muestra en la pantalla.

```
sys.exit(app.exec_())
```

Finalmente, ingresamos al mainloop de la aplicación. El manejo del evento comienza desde este

punto. Mainloop recibe eventos del sistema de ventanas y los envía a los widgets de la aplicación. Mainloop finaliza si llamamos al método `exit()` o el widget principal se destruye. El método `sys.exit()` garantiza una salida limpia. El entorno será informado de cómo terminó la aplicación.

El método `exec_()` tiene un guión bajo. Es porque el exec es una palabra clave de Python. Y así, se `exec_()` su lugar.

Agregar un icono de aplicación

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget
from PyQt5.QtGui import QIcon


class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        self.setGeometry(300, 300, 300, 220)
        self.setWindowTitle('Icon')
        self.setWindowIcon(QIcon('web.png'))

        self.show()


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

Análisis

Argumentos de función en Python

En Python, las funciones definidas por el usuario pueden tomar cuatro tipos diferentes de argumentos.

1. *Argumentos predeterminados:*

- Definición de la función

  ```
  def defaultArg( name, msg = "Hello!"):
  ```

- Llamada de función

  ```
  defaultArg( name)
  ```

2. *Argumentos requeridos:*

- Definición de la función

```
def requiredArg (str,num):
```

- Llamada de función:

```
requiredArg ("Hello",12)
```

3. *Argumentos de palabras clave:*

- Definición de la función

```
def keywordArg( name, role ):
```

- Llamada de función

```
keywordArg( name = "Tom", role = "Manager")
```

o

```
keywordArg( role = "Manager", name = "Tom")
```

4. *Número variable de argumentos:*

- Definición de la función

```
def varlengthArgs(*varargs):
```

- Llamada de función

```
varlengthArgs(30,40,50,60)
```

```
class Example(QWidget):

    def __init__(self):
        super().__init__()
        ...
```

Tres cosas importantes en la programación orientada a objetos son las clases, los datos y los métodos. Aquí creamos una nueva clase llamada `Example` . La clase `Example` hereda de la clase `QWidget` . Esto significa que llamamos a dos constructores: el primero para la clase `Example` y el segundo para la clase heredada. El método `super()` devuelve el objeto principal de la clase `Example` y llamamos a su constructor. La `self` variable se refiere al objeto en sí.

¿Por qué hemos usado `__init__` ?

Mira esto:

```
class A(object):
    def __init__(self):
        self.lst = []

class B(object):
    lst = []
```

y ahora prueba:

```
>>> x = B()
>>> y = B()
>>> x.lst.append(1)
>>> y.lst.append(2)
>>> x.lst
[1, 2]
>>> x.lst is y.lst
True
```

y esto:

```
>>> x = A()
>>> y = A()
>>> x.lst.append(1)
>>> y.lst.append(2)
>>> x.lst
[1]
>>> x.lst is y.lst
False
```

¿Significa esto que x en la clase B se establece antes de la instanciación?

Sí, es un atributo de clase (se comparte entre instancias). Mientras que en la clase A es un atributo de instancia.

```
self.initUI()
```

La creación de la GUI se delega en el método `initUI()` .

```
self.setGeometry(300, 300, 300, 220)
self.setWindowTitle('Icon')
self.setWindowIcon(QIcon('web.png'))
```

Los tres métodos han sido heredados de la clase `QWidget` . El `setGeometry()` hace dos cosas: ubica la ventana en la pantalla y establece su tamaño. Los primeros dos parámetros son las posiciones x e y de la ventana. El tercero es el ancho y el cuarto es la altura de la ventana. De hecho, combina los métodos `resize()` y `move()` en un método. El último método establece el icono de la aplicación. Para ello, hemos creado un objeto `QIcon` . El `QIcon` recibe la ruta de acceso a nuestro icono que se mostrará.

```
if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

La aplicación y los objetos de ejemplo se crean. Se inicia el bucle principal.

Mostrando una sugerencia

```
import sys
from PyQt5.QtWidgets import (QWidget, QToolTip,
    QPushButton, QApplication)
from PyQt5.QtGui import QFont


class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        QToolTip.setFont(QFont('SansSerif', 10))

        self.setToolTip('This is a <b>QWidget</b> widget')

        btn = QPushButton('Button', self)
        btn.setToolTip('This is a <b>QPushButton</b> widget')
        btn.resize(btn.sizeHint())
        btn.move(50, 50)

        self.setGeometry(300, 300, 300, 200)
        self.setWindowTitle('Tooltips')
        self.show()


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

Análisis

```
QToolTip.setFont(QFont('SansSerif', 10))
```

Este método estático establece una fuente utilizada para representar información sobre herramientas. Utilizamos una fuente de 10px SansSerif.

```
self.setToolTip('This is a <b>QWidget</b> widget')
```

Para crear una información sobre herramientas, llamamos al método `setTooltip()` . Podemos utilizar el formato de texto enriquecido.

```
btn = QPushButton('Button', self)
btn.setToolTip('This is a <b>QPushButton</b> widget')
```

Creamos un widget de botón pulsador y configuramos una descripción para él.

```
btn.resize(btn.sizeHint())
btn.move(50, 50)
```

El botón está siendo redimensionado y movido en la ventana. El método `sizeHint()` da un tamaño recomendado para el botón.

Empaquetar su proyecto en excutable / instalador

cx_Freeze - una herramienta puede empaquetar su proyecto en excutable / installer

- después de instalarlo por pip, para empaquetar `demo.py` , necesitamos `setup.py` continuación.

```
import sys
from cx_Freeze import setup, Executable

# Dependencies are automatically detected, but it might need fine tuning.
build_exe_options = {
    "excludes": ["tkinter"],
    "include_files":[('./platforms','./platforms')] # need qwindows.dll for qt5 application
}

# GUI applications require a different base on Windows (the default is for a
# console application).
base = None
if sys.platform == "win32":
    base = "Win32GUI"

setup(  name = "demo",
        version = "0.1",
        description = "demo",
        options = {"build_exe": build_exe_options},
        executables = [Executable("demo.py", base=base)])
```

- luego construir

```
python .\setup.py build
```

- entonces dist

```
python .\setup.py bdist_msi
```

Lea Empezando con pyqt5 en línea: https://riptutorial.com/es/pyqt5/topic/7403/empezando-con-pyqt5

Capítulo 2: Introducción a las barras de progreso

Introducción

Las barras de progreso son una parte integral de la experiencia del usuario y ayudan a los usuarios a tener una idea del tiempo que queda para un proceso determinado que se ejecuta en la GUI. Este tema tratará los aspectos básicos de la implementación de una barra de progreso en su propia aplicación.

Este tema tratará ligeramente sobre QThread y el nuevo mecanismo de señales / ranuras. También se espera un cierto conocimiento básico de los widgets de PyQt5 de los lectores.

Cuando se agregan ejemplos, solo se utilizan los complementos PyQt5 y Python para demostrar la funcionalidad.

Sólo PyQt5

Observaciones

Experimentar con estos ejemplos es la mejor manera de comenzar a aprender.

Examples

Barra de progreso de PyQt básica

Esta es una barra de progreso muy básica que solo usa lo que se necesita como mínimo.

Sería prudente leer todo este ejemplo hasta el final.

```
import sys
import time

from PyQt5.QtWidgets import (QApplication, QDialog,
                             QProgressBar, QPushButton)

TIME_LIMIT = 100

class Actions(QDialog):
    """
    Simple dialog that consists of a Progress Bar and a Button.
    Clicking on the button results in the start of a timer and
    updates the progress bar.
    """
    def __init__(self):
        super().__init__()
        self.initUI()
```

```
    def initUI(self):
        self.setWindowTitle('Progress Bar')
        self.progress = QProgressBar(self)
        self.progress.setGeometry(0, 0, 300, 25)
        self.progress.setMaximum(100)
        self.button = QPushButton('Start', self)
        self.button.move(0, 30)
        self.show()

        self.button.clicked.connect(self.onButtonClick)

    def onButtonClick(self):
        count = 0
        while count < TIME_LIMIT:
            count += 1
            time.sleep(1)
            self.progress.setValue(count)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = Actions()
    sys.exit(app.exec_())
```

La barra de progreso se importa primero `from PyQt5.QtWidgets import QProgressBar`

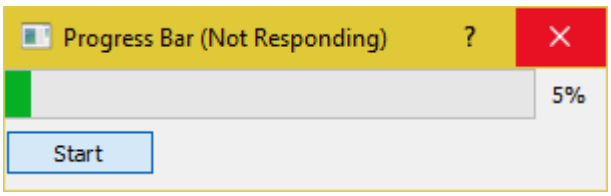
Luego se inicializa como cualquier otro widget en `QtWidgets`

El `self.progress.setGeometry(0, 0, 300, 25)` línea `self.progress.setGeometry(0, 0, 300, 25)` define las posiciones `x,y` en el cuadro de diálogo y el ancho y alto de la barra de progreso.

Luego movemos el botón usando `.move()` en `30px` hacia abajo para que haya un espacio de `5px` entre los dos widgets.

Aquí `self.progress.setValue(count)` se usa para actualizar el progreso. Establecer un valor máximo utilizando `.setMaximum()` también calculará automáticamente los valores para usted. Por ejemplo, si el valor máximo se establece en 50, dado que `TIME_LIMIT` es 100 `TIME_LIMIT` de 0 a 2 a 4 por ciento en lugar de 0 a 1 a 2 cada segundo. También puede establecer un valor mínimo usando `.setMinimum()` forzando que la barra de progreso comience a partir de un valor dado.

La ejecución de este programa producirá una GUI similar a esta.



Como puede ver, la GUI definitivamente se congelará y no responderá hasta que el contador cumpla con la condición `TIME_LIMIT` . Esto se debe a que `time.sleep` hace que el sistema operativo crea que el programa se ha atascado en un bucle infinito.

Qhilo

Entonces, ¿cómo superamos este problema? Podemos usar la clase de subprocesos que

proporciona PyQt5.

```
import sys
import time

from PyQt5.QtCore import QThread, pyqtSignal
from PyQt5.QtWidgets import (QApplication, QDialog,
                             QProgressBar, QPushButton)

TIME_LIMIT = 100

class External(QThread):
    """
    Runs a counter thread.
    """
    countChanged = pyqtSignal(int)

    def run(self):
        count = 0
        while count < TIME_LIMIT:
            count +=1
            time.sleep(1)
            self.countChanged.emit(count)

class Actions(QDialog):
    """
    Simple dialog that consists of a Progress Bar and a Button.
    Clicking on the button results in the start of a timer and
    updates the progress bar.
    """
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle('Progress Bar')
        self.progress = QProgressBar(self)
        self.progress.setGeometry(0, 0, 300, 25)
        self.progress.setMaximum(100)
        self.button = QPushButton('Start', self)
        self.button.move(0, 30)
        self.show()

        self.button.clicked.connect(self.onButtonClick)

    def onButtonClick(self):
        self.calc = External()
        self.calc.countChanged.connect(self.onCountChanged)
        self.calc.start()

    def onCountChanged(self, value):
        self.progress.setValue(value)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = Actions()
    sys.exit(app.exec_())
```

Vamos a desglosar estas modificaciones.

```
from PyQt5.QtCore import QThread, pyqtSignal
```

Esta línea importa `Qthread` que es una implementación de `PyQt5` para dividir y ejecutar algunas partes (por ejemplo: funciones, clases) de un programa en segundo plano (también conocido como multihilo). Estas partes también se llaman hilos. `PyQt5` defecto, todos los programas de `PyQt5` tienen un subproceso principal y los otros (subprocesos de trabajo) se utilizan para descargar más tiempo y procesar tareas intensivas en segundo plano mientras se mantiene el funcionamiento del programa principal.

La segunda importación `pyqtSignal` se utiliza para enviar datos (señales) entre los procesos de trabajo y los subprocesos principales. En este caso, lo utilizaremos para indicar al subproceso principal que actualice la barra de progreso.

Ahora hemos movido el bucle while para el contador a una clase separada llamada `External` .

```
class External(QThread):
    """
    Runs a counter thread.
    """
    countChanged = pyqtSignal(int)

    def run(self):
        count = 0
        while count < TIME_LIMIT:
            count +=1
            time.sleep(1)
            self.countChanged.emit(count)
```

`QThread` esencialmente estamos convirtiendo `External` en una clase que puede ejecutarse en un hilo separado. Los hilos también pueden iniciarse o detenerse en cualquier momento, lo que se suma a sus beneficios.

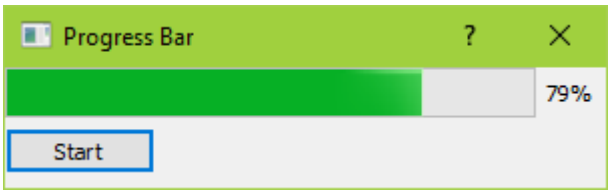
Aquí `countChanged` es el progreso actual y `pyqtSignal(int)` le dice al subproceso de trabajo que la señal que se envía es de tipo `int` . Mientras que `self.countChanged.emit(count)` simplemente envía la señal a cualquier conexión en el subproceso principal (normalmente, también se puede usar para comunicarse con otros subprocesos de trabajo).

```
def onButtonClick(self):
        self.calc = External()
        self.calc.countChanged.connect(self.onCountChanged)
        self.calc.start()

def onCountChanged(self, value):
    self.progress.setValue(value)
```

Cuando se hace clic en el botón, `self.onButtonClick` se ejecutará y también iniciará el hilo. El hilo comienza con `.start()` . También se debe tener en cuenta que conectamos la señal `self.calc.countChanged` que creamos anteriormente al método utilizado para actualizar el valor de la barra de progreso. Cada vez que `External::run::count` se actualiza, el valor `int` también se envía a `onCountChanged` .

Así es como podría verse la GUI después de realizar estos cambios.



También debe sentirse mucho más sensible y no se congelará.

Lea Introducción a las barras de progreso en línea:
https://riptutorial.com/es/pyqt5/topic/9544/introduccion-a-las-barras-de-progreso

Creditos

S. No	Capítulos	Contributors
1	Empezando con pyqt5	Ansh Kumar, Community, ekhumoro, suiwenfeng
2	Introducción a las barras de progreso	daegontaven